SYNOPSIS

SKYLON

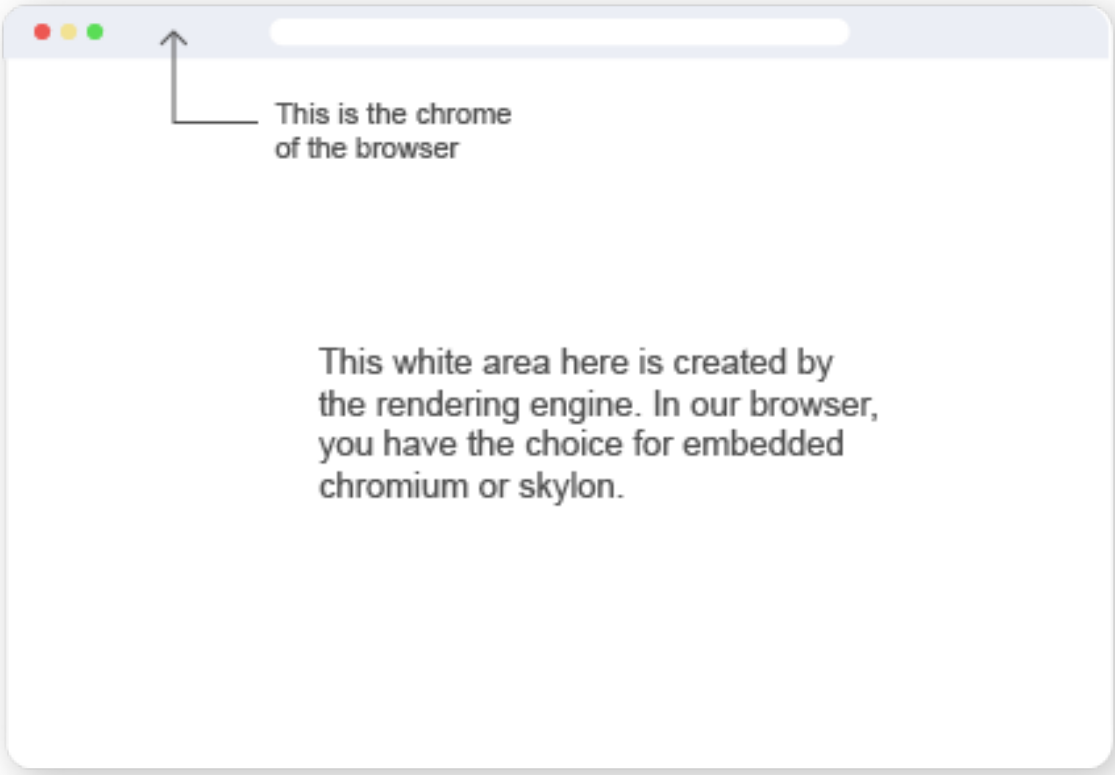
BRINGING PYTHON TO THE WEB

BY ~ SUJAL SINGH

INTRODUCTION

The **World Wide Web**, developed by **Tim Berners Lee** in 1989 while working at **CERN**. Initially the web was meant to share documents between scientists of various universities. One couldn't have imagined how much the web has evolved today. A software that is used to interact with the web is the **web browser**.

Web browsers are complex machinery, one could argue modern web browser's development is more complex than an entire operating system. **Chromium** an open-source browser, has 35 million lines of code in it, compared with **Linux** kernel having 15 million lines combined with **GNU** operating system having another 14 adding up to only 29 million lines.



A web browser has primarily two components, one being the **chrome** of the browser that is tiny search bar and other related GUI

OBJECTIVE

The objective of this project is to create a web browser which replaces JavaScript with Python (it still maintains JavaScript support through CEF).

The objective might seem simple but the applications of such a project are quite a few. One could utilize this as a GUI framework for python, like Electron JS for JavaScript. Such an application allows a python developer to write code once and package and distribute the same code base to all desktop platforms and on the web too. The set of all use cases for the web is a subset of the use cases of our application.

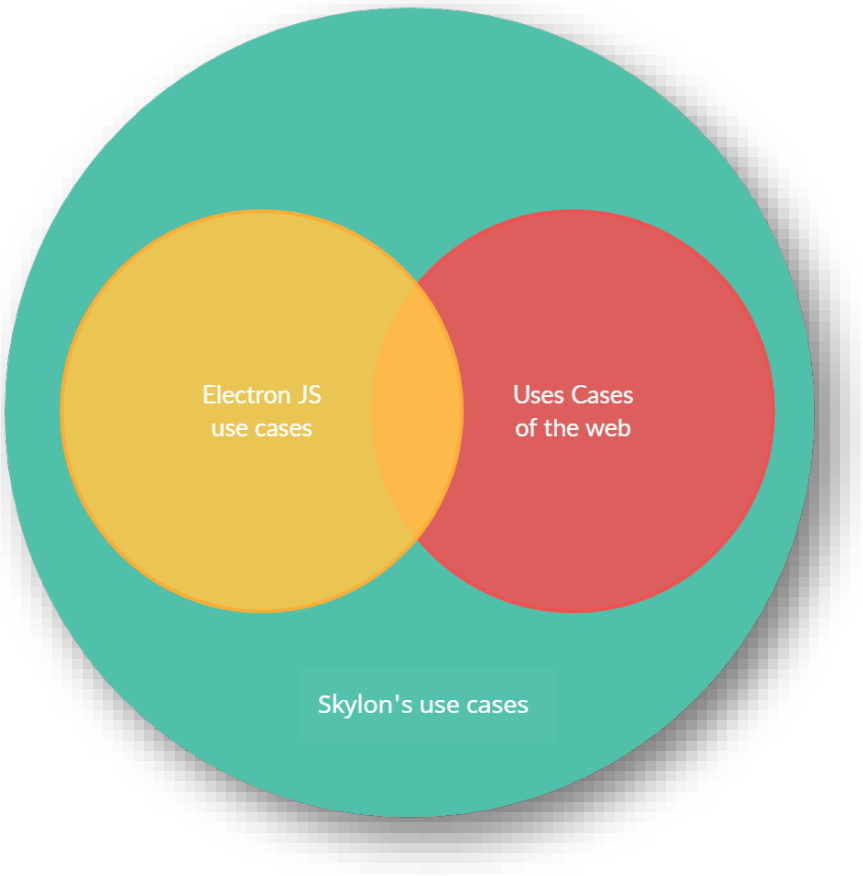


Figure 1: Use cases of Skylon

WORKING

Our browser has two operation modes one for chromium embedded and one for Skylon. The browser's launch process is explained with a flow chart below:

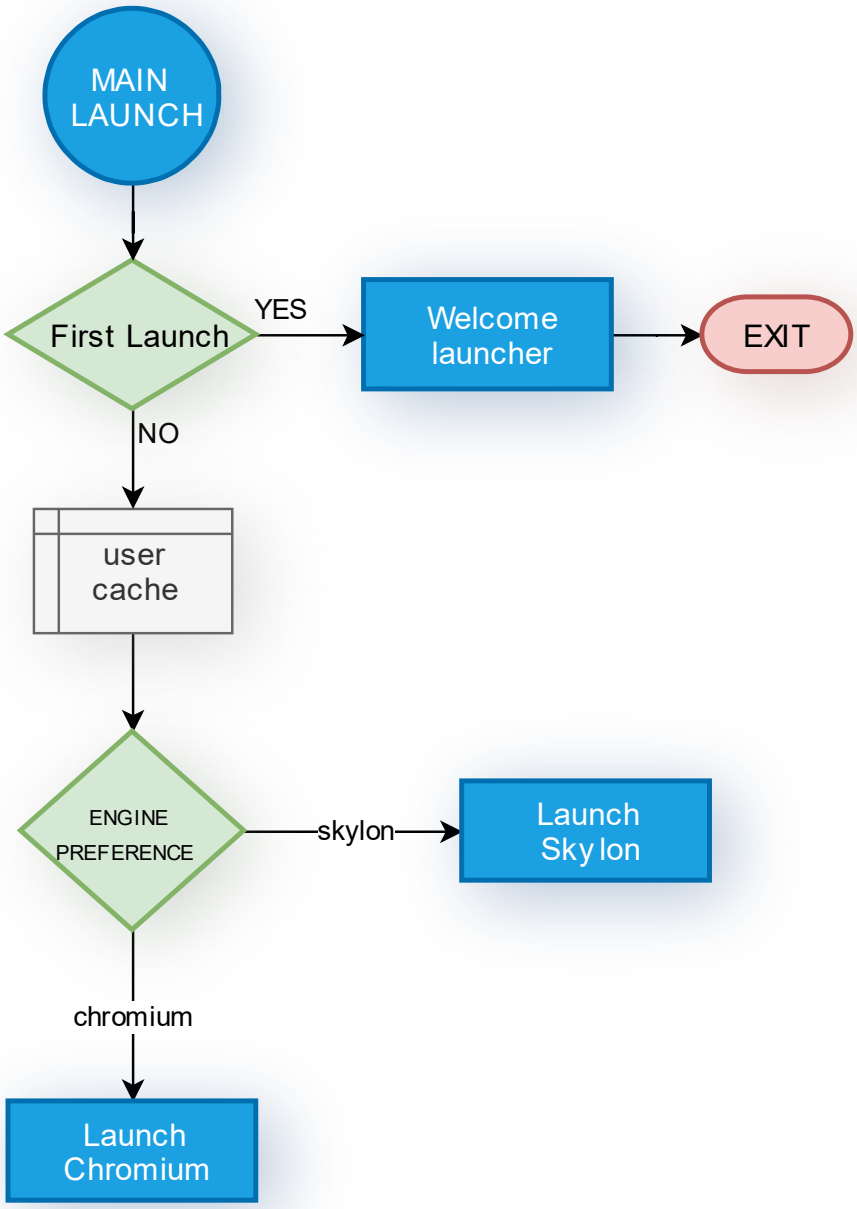


Figure 2: Main Launcher (entry-point)

The welcome launcher is a window created by CEF Python; it displays the webpage https://sujalsinghx86.github.io/school-project-web/ as the GUI. The launcher also exposes some python functions to the website using CEF Python to get data entered into the input boxes. Then it connects to a real MySQL server hosted on AWS here, storing the user's browser preference, and if it already exists does not ask for the preference. It also caches the data locally using a binary file for quicker access.

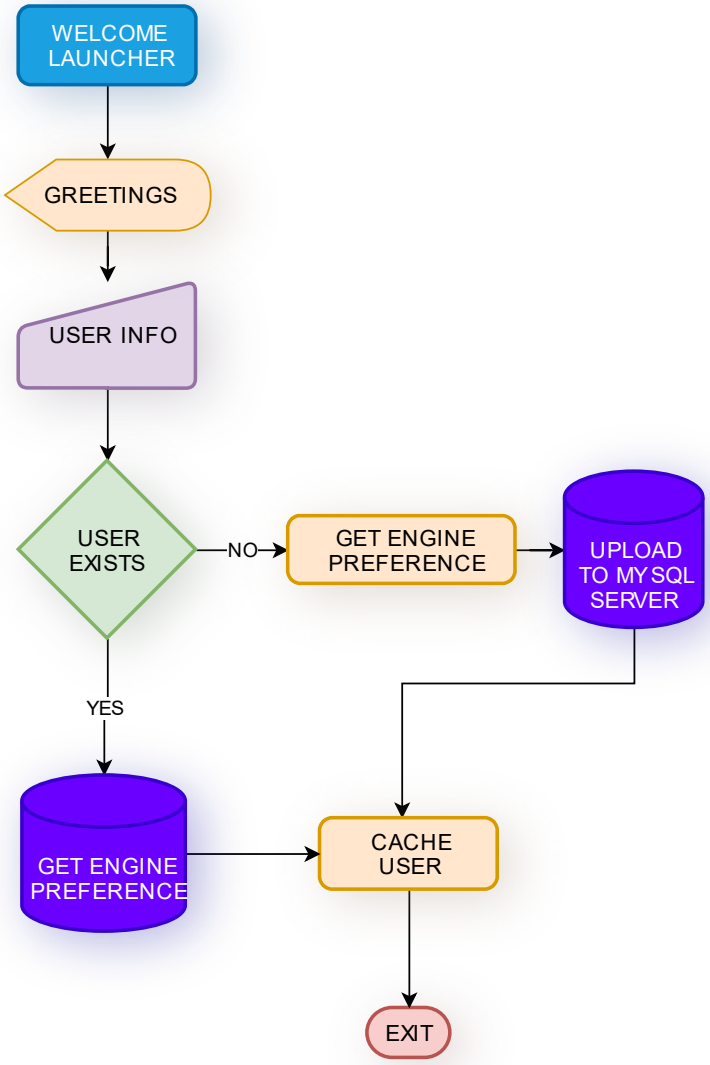


Figure 3: Welcome Launcher

Launching Skylon and Chromium are simple enough procedures, so no flow chart is required for them.

How Skylon works?

As mentioned earlier, browsers are complex and so here we explain only the features we have implemented in our browser.

Before one can understand how browsers work, one should be aware of the existence of **web standards**. Web Standards are in a way the **laws of the web**. All major browsers conform to these standards to avoid ambiguity in the web. The organization called **W3C (World Wide Web Consortium)**, founded by the creator of the web Tim Berners Lee maintains these standards. Although, HTML standards are being maintained by a separated body called **WHATWG**. Our browser also tries to conform to these standards to the best of our abilities.

Having the knowledge from these standards one can now figure out how to implement a rendering engine for a browser, our browser's working can be demonstrated with some flowcharts:

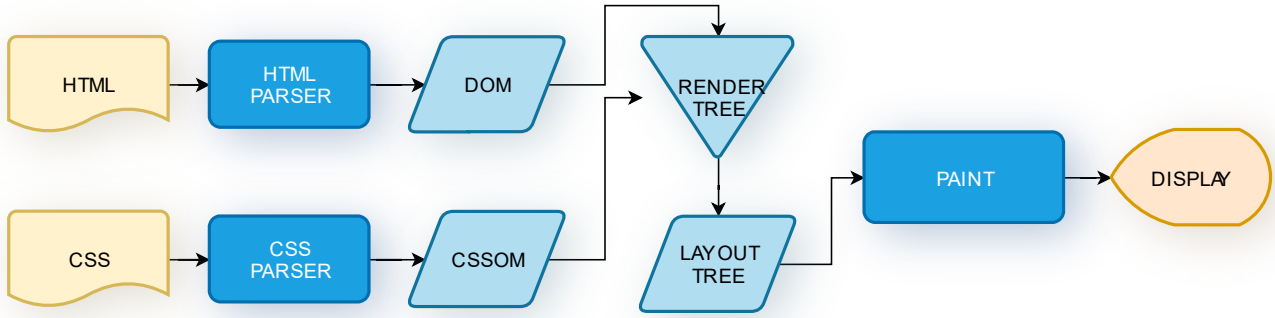
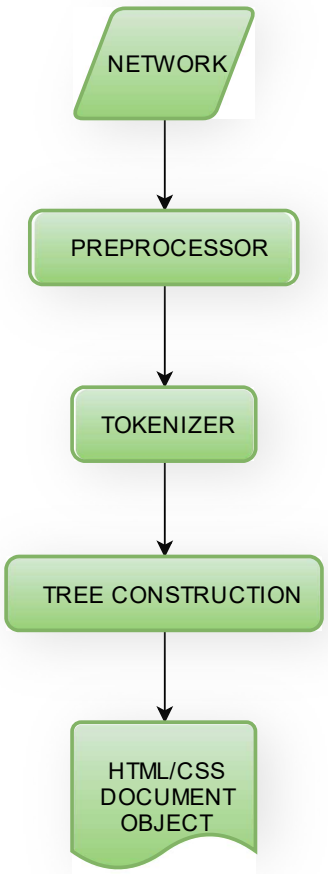


Figure 4: Rendering Engines

HTML documents are parsed into DOM objects and similar process happens for CSS; the outputs of these stages are combined to create a render tree. The render tree consists of only those elements that are to be displayed, for example an element with the display property set to none will not appear in the render tree. The render tree is then converted into a layout tree in which all elements know where they are to be positioned, how they look (color, dimensions, etc.). The painting process utilizes this tree to finally tell the GUI framework (PyQT5 in our project) to display items according to the tree.

Parsing



Requested document is fetched through the internet.

Preprocessor removes unnecessary characters from fetched document.

The tokenizer breaks the input stream into its lexical components. For example, <html> into a tag element.

The tree construction stage takes in input from the tokenizer and makes a tree of parent and child nodes.

The above tree is contained in a Document object, that is further rendered into a web page.

Figure 5: Parsing Flow

Tokenizing

The web standards model the tokenizer as a state machine. A state machine processes inputs according to the rules of its current state and inputs can change its state. Here is a tinier version of the tokenizer's state machine diagram:

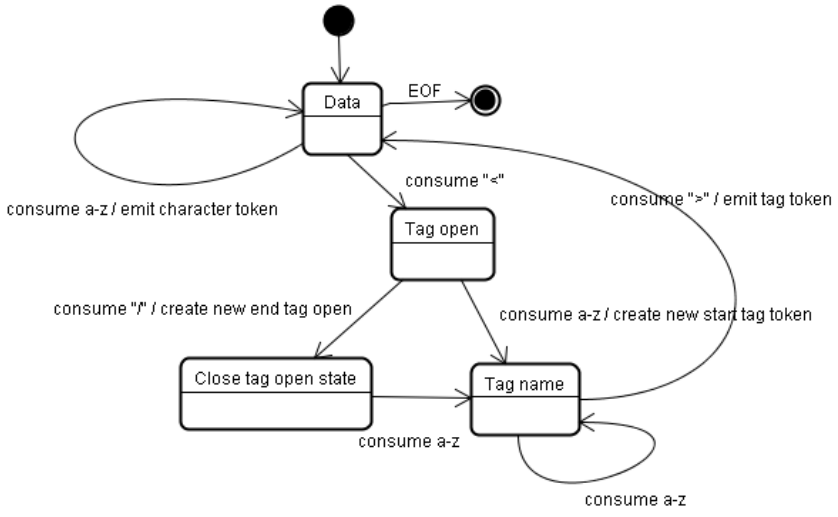


Figure 6: Tokenizer State Machine Diagram